CS 505 Ex6

Prashant Ravi — ravi18@purdue.edu

March 27, 2016

1 Step1

Two precautions are in place to avoid a scenario where the message doesn't reach before the timeout. The first is that once the leader program starts on any machine it starts the receiver thread thread that is set up to receive datagram packets waits 2000 milliseconds before sending out its proposal value. By this time, we are sure that each process would've started its receiving thread and the algorithm can progress. The second precaution in place is that when a process sends out a proposal the receiving process, if hasn't received a value from sending process in the past responds to the sender with its proposal value. This way, if the first process had sent the proposal out to second process initially, but for some reason the receiving process was being slow and didn't have its receiver thread running at the time and ended up not adding sender's value to its set, then when the second process does eventually start up, on sending its proposal value to the first process it would not only propose its value but also retrieve the senders original proposal. The second precaution in practice hasn't been necessary as the processes started correctly within the 2000 millisecond window. However, in the circumstance the above described issue does arise, it would be circumvented with the additional complexity of $O(n^2)$ extra messages being passed in the worst case. This is for the condition that first process sends but no one else receives, then second process sends but only first process receives, continuing forth till n processes.

The above feature of sending back a proposal on having not correctly delivered the proposal the first time, has been realized with the help of serializing the classes, enclosing the proposal value to denote specific tasks to be executed on the deserialized class, based on the class type, on the receiving end. For instance, ClientReply signifies a message proposal that doesn't expect the receiver to send back its original proposal. However, by enclosing a proposal in ClientPropose the sender expects that the receiver of this message will also send back its proposal value.

2 Step 2

To realize this part all the changes that had to be made were that the processes would propose their own ids, whichever correct process had the largest id would win, and be heralded the leader.

3 Step 3

The algorithm for the leader election problem in a synchronous message passing setting with crash failures is as follows:

```
tResilient-Consensus(id, f):
    // Each Process P_i where i ={1,n} executes the code for up to f crashes.
    // f: maximum number of crashes in the system.
    // id: the id of the host code is running on.

    decided: integer; // leader id at the end of f+1 rounds,
    decided = id; // initially set decided to id of process at hand
    decidedChanged: boolean; // Initially false;
    Initially V_i is {}
    for round 1 to f+1 do:
        if(decidedChanged):
                decidedChanged = false;
                Broadcast decided to all
        Receive V_j from all p_j where j is 1 to n
        V_i = V_i U V_j for all j 1 to n
        y_i = max(V_i) // decided value.
        if(y_i > decided)
            decidedChanged = true;
            decided = y_i
    output decided as the consensus value
```

3.1 Proof of correctness

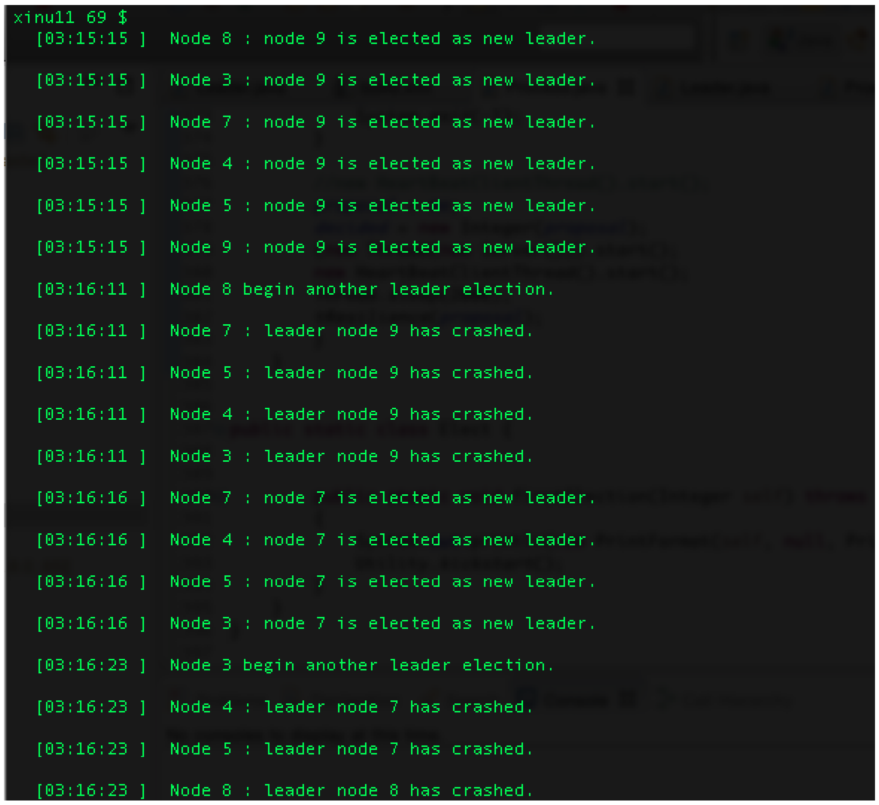
For the f+1 rounds, there must have been at least one round r where no process that has been correct so far, would've failed. So in this round the all correct processes so far would be successful in broadcasting their message to the other processes. And all the receiving processes then go ahead to take the minimum of these processes' proposed values. The worst case scenario is for when in a round if the largest value is with a process when it sends it to a second process before failing, now the largest values is with second process, and it sends it right before failing in the subsequent round, this could continue until f+1 rounds Assume that the largest value was only received in the last round then there would be a chain of f+1 processes that transferred the value from one of the process to the next.If all the processes in the chain are faulty then we violate the claim that there are at most f faulty processes. If any of the process was non-faulty in this chain then it would have been successful in sending the consensus value to all the other processes in that round.Hence, at most f+1 rounds are required for the processes to come to consensus. So, after f+1 rounds we can be sure the local decided value for each process is the consensus value of the system of processes. Hence, agreement condition is satisified. Validity and termination are also observed to be satisified.

Termination :

4 Step 4

To detect if a process has crashed or is still alive I have employed a heartbeat mechanism. Basically, every 3 seconds in an all -to -all scheme every process broadcasts a hearbeat to every other process, one by one, and expects that the receiving process heart beat back to the sending process within two round trip times, set to be 100ms in our case. If any process doesn't respond to a correct process's heart beat, within the above stated time, it is deemed as crashed. Two classes are created to signify a hear beat and hear beat back. And on deserializing the response on the receiving thread of a process it is enacted upon differently. For instance, a heart beat back class would just set the flag on a process as "received" for the sending process, whereas a heat beat class would also expect the receiving process to send back a heart beat back serialized object as a datagram packet. We utilize this especially to detect if the leader process is dead. A pseudo random generator is employed and the elected leader is crashed for testing purposed with prob 1/60 each time a heart beat is sent out to the other processes. The Kill class would basically do the necessary actions to kill the leader process, if and when the conditions in the pseudo random generator are met. At this point, the first process to detect the leader has having crashed starts a new leader election algorithm and sends out this message to the other processes so they too prepare to start a new leader election iteration.

For the leader reelection problem this was my result.



```
xinu11 69 $
  [03:15:15 ]  Node 8 : node 9 is elected as new leader.

  [03:15:15 ]  Node 3 : node 9 is elected as new leader.

  [03:15:15 ]  Node 7 : node 9 is elected as new leader.

  [03:15:15 ]  Node 4 : node 9 is elected as new leader.

  [03:15:15 ]  Node 5 : node 9 is elected as new leader.

  [03:15:15 ]  Node 9 : node 9 is elected as new leader.

  [03:16:11 ]  Node 8 begin another leader election.

  [03:16:11 ]  Node 7 : leader node 9 has crashed.

  [03:16:11 ]  Node 5 : leader node 9 has crashed.

  [03:16:11 ]  Node 4 : leader node 9 has crashed.

  [03:16:11 ]  Node 3 : leader node 9 has crashed.

  [03:16:16 ]  Node 7 : node 7 is elected as new leader.

  [03:16:16 ]  Node 4 : node 7 is elected as new leader.

  [03:16:16 ]  Node 5 : node 7 is elected as new leader.

  [03:16:16 ]  Node 3 : node 7 is elected as new leader.

  [03:16:23 ]  Node 3 begin another leader election.

  [03:16:23 ]  Node 4 : leader node 7 has crashed.

  [03:16:23 ]  Node 5 : leader node 7 has crashed.

  [03:16:23 ]  Node 8 : leader node 8 has crashed.
```